Programming in XL (part 2)

Queries & Co.

Katarína Streit (Smoleňová)

ksmolen@gwdg.de
Departement Ecoinformatics, Biometry and Forest Growth
Georg-August-University Göttingen

September 17, 2013 / Prague

"Modelling of Ecosystems by Tools from Computer Science"

Outline

- Execution rules
- Queries
- Aggregating operators
- Derived relations

Execution rules

- Used to execute imperative statements for searched organs or update their attribute values, without changes in the (graph) structure
- XL syntax: `:::>`

```
ModuleName ::> { imperative code; }
```

Example:

```
l:Leaf ::> { l.photosynthesis(); }
```

L-system rule

```
Internode(length, radius) ==>
    Internode(length+0.1, radius+0.01)
;
```

Execution rule

```
i:Internode ::> {
    i[length] += 0.1;
    i[radius] += 0.01;
}
```

Deferred assignment operator

- Deferred assignments are not executed immediately, but at some later point (final application of a parallel production)
- Parallel execution of assignments
- XL syntax: **:=**
- Quasi-parallel assignment to the variables x and y:

```
x := f(x, y);
y := g(x, y);
```

Execution rule

```
i:Internode ::> {
    i[length] :+= 0.1;
    i[radius] :+= 0.01;
}
```

Graph query

- To analyse the actual structure (graph), i.e. search for a specific pattern
- XL syntax: enclosed in asterisked parenthesis `(* *)`
- The elements are given in their expected order, e.g.: `(* A A B *)` searches for a subgraph which consists of a sequence of nodes of the types `A A B`, connected by successor edges

Examples

- Find all internodes, and print them out

```
println((* Internode *));
```

- Find all newly created internodes (with age 0)

```
(* i:Internode, (i[age] == 0) *)
```

- Search for all internodes with diameter > 0.01

```
(* i:Internode, (i[diameter] > 0.01) *)
```

- Find all pairs with distance < 1

```
(* f:F, g:F, ((f != g) && (distance(f, g) < 1)) *)
```

- Find all nodes B, connected to A with a branching edge

```
(* A +> B *)
```

Aggregate methods

- Collect multiple values and return one single value as result
- Standard aggregate operations:
  `count`, **`sum`**, **`empty`**, **`exist`**, **`forall`**, **`first`**, **`last`**, **`max`**, **`min`**, **`mean`**, **`selectRandomly`**, **`selectWhereMin`**, **`selectWhereMax`**, ...
- Can be applied to set-valued results of a query

```
count((* Leaf *))
first((* l:Leaf, (l[order] == 1 && l[rank] == 1) *))
selectRandomly((* F *))
selectWhereMax((* f:F *), (f[diameter]))
```

Examples

- Count all segments F, longer than 1

```
count((* f:F, (f[length] > 1) *))
```

- Search for all leaves and sum up their surface areas

```
sum((* Leaf *)[area])
sum((* Leaf *).area)     // alternative
```

- Sum up potentional growth rate of all growing leaves

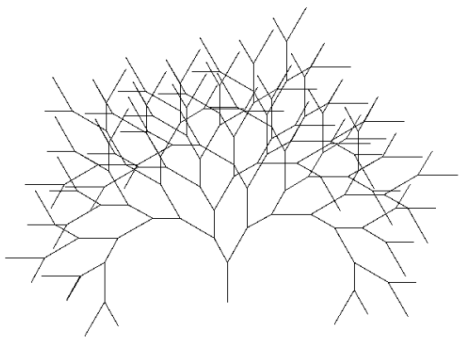
```
sum((* l:Leaf, (l.isGrowing()) *).pgr())
```

Check distance example

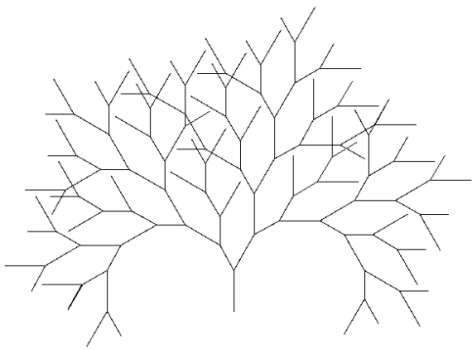
- Binary tree, growth shall start only if there is enough distance to other F objects

```
module A(float len);
// inside the init method:
Axiom ==> F(1) [ RU(-30) A(0.7) ] RU(30) A(1);
// inside the run method:
a:A(s) ==>
    if (forall(distance(a, (* F *)) > 0.6)) (
            RH(180) F(s) [ RU(-30) A(0.7) ] RU(30) A(1)
    )
;
```

without the „if“ condition



with the „if“ condition



A simple model of overshadowing

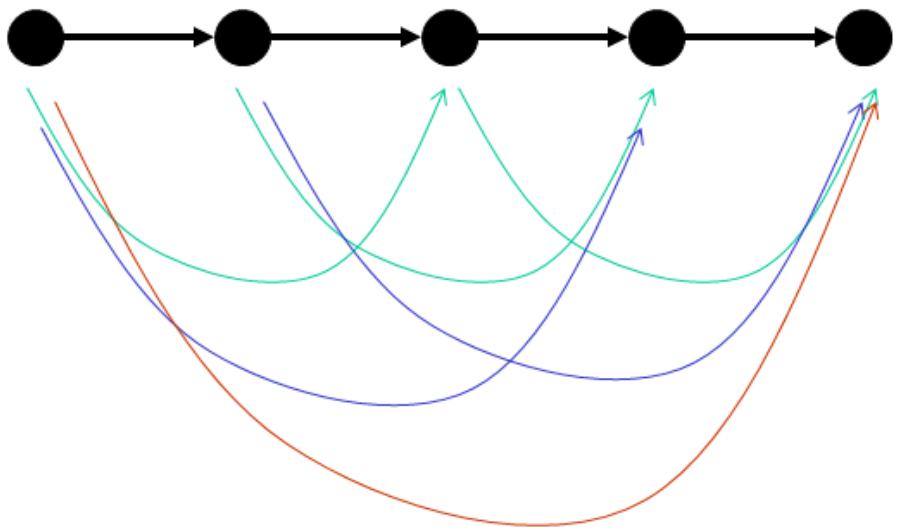
- Model approach (strongly simplifying): overshadowing of an object occurs when there are further objects in an imagined cone with its apex in the object, opened into z direction (to the sky)
- Using a query referring to a geometric region in space

```
Vector3d z = new Vector3d(0, 0, 1);

x:TBud(t), (t >= 0 &&
    empty((* s:Segment, (s in cone(x, z, 45)) *))) ==>

    // grow
    ...

;
```

sm09_e42.rgg

Derived relations

- Relation between nodes connected by several edges (one after the other) of the same type
  example: find all descendants of some given node which are of type Internode
- “Transitive hull” of the original relation (edge)



XL notation

- 1-to-*n* repetitions: **+**

```
A (-edgetype->)+ B
```

- 0-to-*n* repetitions: *****

```
A (-edgetype->)* B
```

- *Minimal elements* (stop searching once a match has been found):

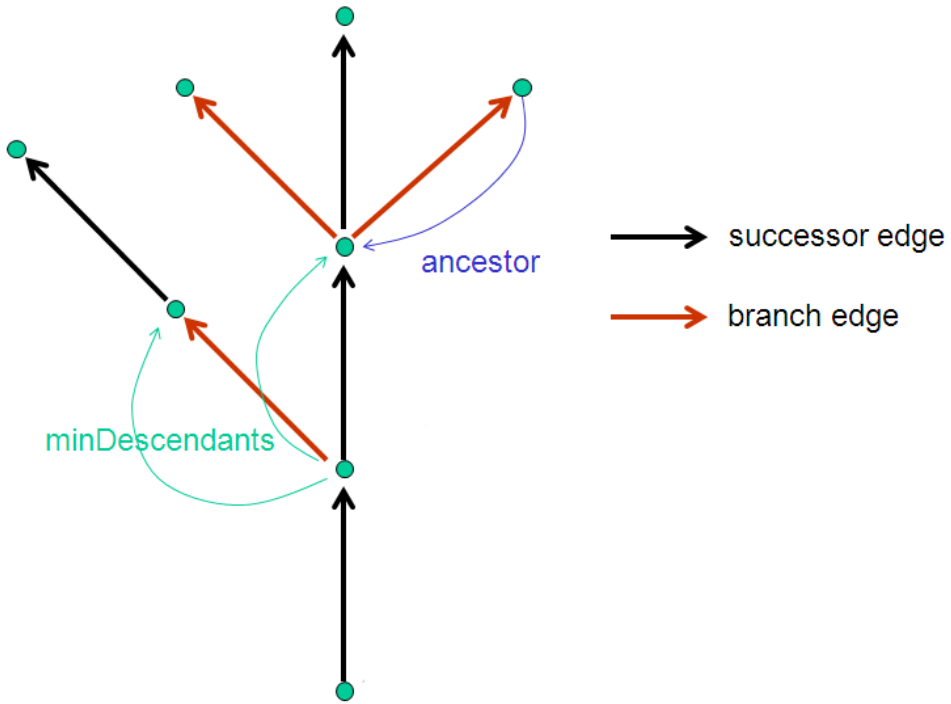
```
A (-edgetype->)+ :(B)
A (-edgetype->)* :(B)
```

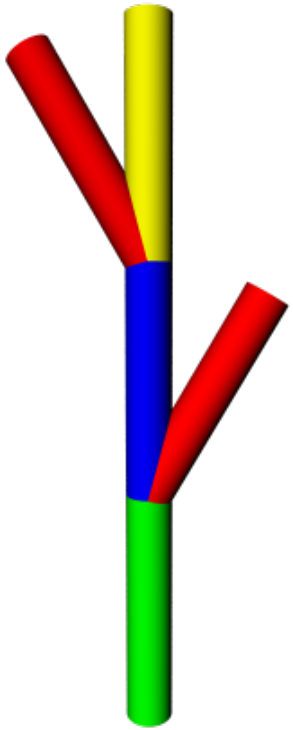
Examples

- Find all/the first internode connected to the leaf

```
(* Leaf (<--)+ Internode *)
(* Leaf (<--)+ :(Internode) *)
```

`ancestor` - nearest preceding node of a certain node type
`minDescendants` - nearest successors of a certain node type (nodes of other types are skipped)
`descendants` - all successors of a certain node type

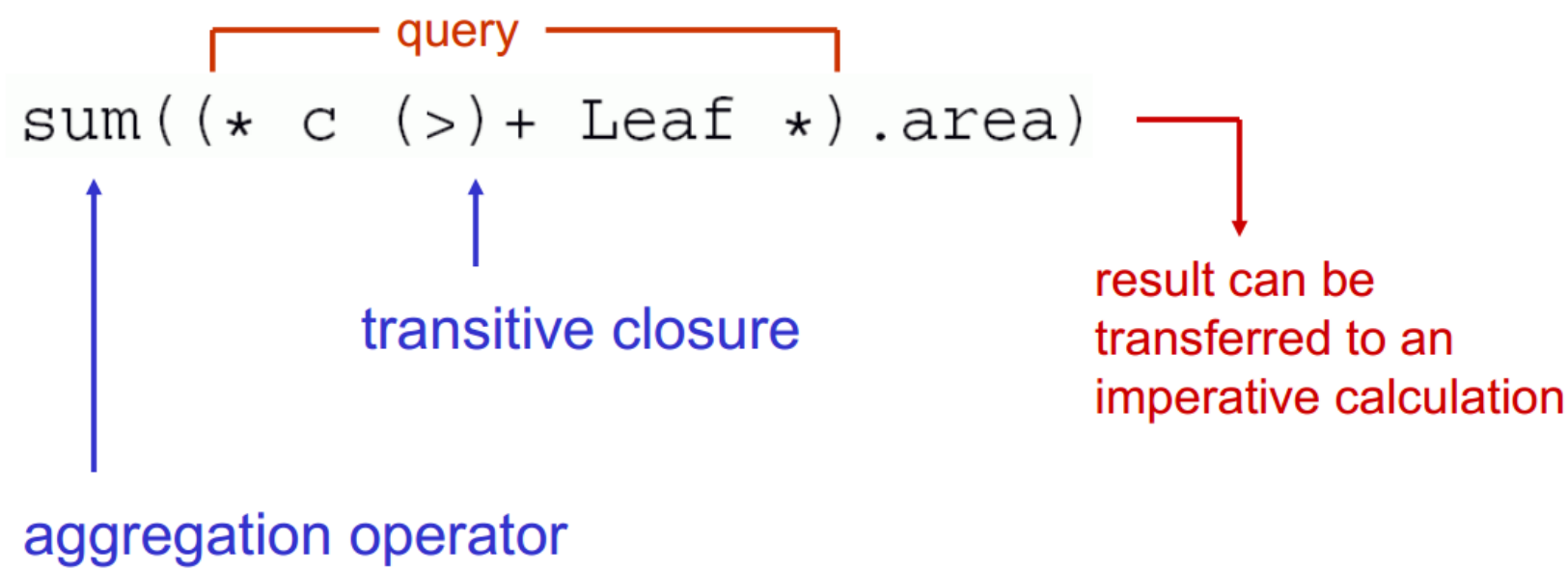




```
// all modules extend the module Internode
Axiom ==>
    InternodeFirst [RU(30) BranchFirst]
    InternodeSecond [RU(-30) BranchSecond]
    InternodeThird
;
// queries and their outputs:
(* InternodeFirst -minDescendants-> Internode *)
    BranchFirst
    InternodeSecond

(* InternodeFirst -descendants-> Internode *)
    InternodeSecond
    InternodeThird
    BranchSecond
    BranchFirst

(* InternodeThird -ancestor-> Internode *)
    InternodeSecond

(* InternodeThird (-ancestor->)+ Internode *)
    InternodeSecond
    InternodeFirst
```

Summary

- Provide possibilities to connect structure and function
- Example: search for all leaves which are successors of node **c** and sum up their surface areas



Thank you for your attention!